A Deep Neural Network Architecture for Decoding Linear Codes Based on the Parity Check Matrix

Samira Malek, Saber Salehkaleybar, Arash Amini, *Senior Member, IEEE*

***Abstract*—Belief propagation is a well-studied decoding technique for linear error correction codes. The method involves the Tanner graph of the code and achieves a near optimal performance under certain settings. In this paper, we propose a new neural network architecture for decoding a linear code inspired by the belief propagation technique applied to the parity check matrix of the code. In particular, our method involves element-wise multiplications instead of matrix multiplications, which substantially reduces the computational complexity of the method and in turn, improves its scalability. Our simulation results on BCH, LDPC, and Polar codes confirm that the proposed method, in addition to the reduced computational burden, outperforms the state of the art methods in terms of the bit error rate.**

***Index Terms*—Belief Propagation algorithm, deep neural network, linear codes, parity check matrix.**

I. Introduction

ERROR correction coding is a well-studied technique to improve the communication throughput along a given channel. Although random codes are known to achieve near optimal performances, decoding a code with no specific structure is computationally impractical. As a result, the challenge of code design is mainly restricted by the available decoding techniques. The message passing approaches such as belief propagation (BP) are among the most successful decoding techniques. The main idea is to form a Tanner graph and iteratively propagate information about the likelihood of the uncoded bits (messages) along the edges of the graph. As the structure of the messages are fixed, these approaches are applicable to almost all codes. It is now interesting to see if one can achieve a superior performance by tailoring almost the same decoding structure to a specific code.

Due to the advances in neural networks, the concept of training a network for decoding error correcting codes is recently gaining more attention in the research world. In [1], a kind of fully connected neural network was designed for decoding short polar and random codes with length 16. Indeed, the method was based on recasting the decoding problem as a classification problem; as the training dataset in a classification problem shall fairly cover the whole space, the size of the dataset for the neural network decoder needs to scale exponentially in terms of the code length, which practically limits the length to small numbers such as 16. Again restricted by the length, in [2], neural networks were used in sub-blocks of a decoder. Specifically, the performance of the standard decoder for Polar codes [3] was improved with this technique.

More recently, a new kind of neural network based on BP was proposed in [4], which preserves the symmetry condition of the BP algorithm [5]; the latter symmetry condition implies that the performance of the network becomes independent of the original codeword. Thus, a training dataset with only a single codeword (e.g., the all-zero vector) and its noisy versions suffices. Besides, the proposed architecture is such that if all weights are set to one (no learning), the network becomes equivalent to the conventional BP decoder. Hence, by optimizing the weights (the training phase), we can only improve the BP performance. A similar approach was applied to the BP decoder adapted to polar codes in [6] with improved hardware implementation. [7] studied the implementation of the min-sum algorithm within this framework and in [8], the quantization step of the min-sum algorithm was improved using neural networks. By adding extra variable layers to the network proposed in [4] without violating the BP symmetry condition, another model was introduced in [9] which had a superior performance under BCH codes. The use of hypernetworks was first investigated in [10], where a network was employed to tune the weights of another network that combines messages coming from check nodes. This technique showed improved performances for a number of LDPC, BCH and polar codes. An extension of this approach has been applied to the modified BP decoder [3] for polar codes in [11].

A recurrent neural network (RNN) was applied to convolutional and Turbo codes in [12]. It is observed that the performance of the RNN structure is very close to the Viterbi and BCJR decoders. In [13], feed-forward neural networks were used for both channel coding and modulation; in addition, a joint turbo autoencoder was trained [14]. The overall performance suggests that this method is a fair alternative to using modern codes; also, the learned modulation blocks have meaningful descriptions.

By focusing on LDPC codes, reinforcement learning was used for decoding codes with sparse factor graphs [15]. While there is no ideal decoder for topological codes in quantum, recently, a BP neural network decoder has been proposed for decoding 2D topological codes with large distances [16].

In this paper, we introduce a new neural network architecture based on the BP algorithm which also preserves the symmetry condition. Although the BP algorithm traditionally runs on the Tanner graph, we utilize a matrix form representation of it in designing our network architecture. The proposed solution not only reduces the overall computational load, but

The authors are with the Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran. E-mails: Maalek.Samira@ee.sharif.edu, {saleh, aamini}@sharif.edu.

also achieves a superior performance in terms of the bit error rate (BER).

The rest of this paper is organized as follows. The Tanner graph and its interplay with the parity check matrix is described in Section II; we, then, continue this section by explaining the standard BP algorithm. In Section III, we first describe how the standard BP algorithm can be applied to the parity check matrix. Next, we propose a neural network structure based on the latter BP interpretation. Performance evaluation of the proposed method and its comparison with the existing approaches are provided in Section IV. Finally, we conclude the paper in Section V.

II. Background

In this section, we first introduce the Tanner graph and review the basics of the conventional BP algorithm.

A. Tanner Graph

For a linear binary code with block size n, where k bits are reserved for the information bits and the remaining $n-k$ bits for parity, the parity check matrix $\mathbf{H}$ of the code has n columns and $n-k$ rows. The non-zero elements in the ith row of $\mathbf{H}$ represent the bits contributing in the ith parity check equation. As an example, the parity check matrix for the $(7,4)$ Hamming code (which is also a BCH code) is:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

The Tanner graph is a bipartite graph associated with the parity check matrix [17]. This graph consists of n variable and $(n-k)$ check nodes corresponding to the columns and rows of the parity check matrix. The jth variable node is connected to the ith check node if and only if the (i,j) element in $\mathbf{H}$ is non-zero.

B. Belief Propagation Algorithm

The belief propagation algorithm is an iterative scheme of propagating messages between the connected variable and check nodes in the Tanner graph [18]. The goal is to obtain the marginal distribution of each bit at the end of the iterations.

For a memory-less channel, the input to the BP algorithm is the log-liklihood ratio (LLR) of each bit based on the observed noisy vector (channel output), which is given by:

$$\mu_v = \log\left(\frac{\mathbb{P}(x_v = 1 | y_v)}{\mathbb{P}(x_v = 0 | y_v)}\right). \tag{1}$$

Here, x_v stands for an original bit (vth bit in the codeword of length n) before transmission through the channel, while y_v represents the corresponding noisy observation at the output of the channel. Each iteration of the BP algorithm has two phases: in the first phase, based on the initial LLR values or the messages received at the end of the previous iteration, variable nodes create and send messages to check nodes. In the second phase, by combining the received messages, each check node sends back a message to its neighboring variable nodes.

To describe the messages, let us denote the message sent from variable node v to check node c in the first phase of the lth iteration by $\mu_{v,c}^{(l)}$. Similarly, let $\mu_{c,v}^{(l)}$ be the message sent from check node c to variable node v in the second phase of the lth iteration. With these notations, the messages are formed as:

$$\mu_{v,c}^{(l)} = \mu_v + \sum_{c' \in C_v \setminus \{c\}} \mu_{c',v}^{(l-1)}, \tag{2}$$

and

$$\mu_{c,v}^{(l)} = 2\text{Arctanh}\left(\prod_{v' \in V_c \setminus \{v\}} \tanh\left(\frac{\mu_{v',c}^{(l)}}{2}\right)\right). \tag{3}$$

Here, C_v identifies all the check nodes incident to the variable node v; similarly, V_c represents all the variable nodes incident to the check node c. By convention, $\mu_{c,v}^{(-1)}$ equals 0 in $l = 0$.

Finally, the marginalization of the messages from check nodes in the lth iteration (usually, the last iteration) are computed by:

$$o_v^{(l)} = \mu_v + \sum_{c' \in C_v} \mu_{c',v}^{(l)}, \tag{4}$$

III. Proposed Method

As mentioned, the standard implementation of the BP algorithm is based on the Tanner graph. Herein, we provide a matrix form representation of the BP algorithm that directly takes into account the parity check matrix. Based on this new representation, we introduce a neural network for the purpose of decoding.

A. Matrix Form Representation of the BP Algorithm

The BP algorithm is commonly explained over the Tanner graph. In particular, messages are sent over the edges of the Tanner graph between variable and check nodes. Moreover, each non-zero entry in the parity check matrix corresponds to an edge in the Tanner graph. By replacing these non-zero entries with messages, we can have a matrix form representation of the BP algorithm. The details of this new representation is given in the sequel.

We replace the input vector $\boldsymbol{\mu} = [\mu_1, \cdots, \mu_n]$ with matrix $\mathbf{M}$ of size $(n-k)\times n$ such that: $[\mathbf{M}]_i = \boldsymbol{\mu} \circ [\mathbf{H}]_i$ where $[\mathbf{A}]_i$ denotes the i-th row of a matrix $\mathbf{A}$ and $\mathbf{A} \circ \mathbf{B}$ returns the element-wise product of the two matrices $\mathbf{A}$ and $\mathbf{B}$.

Similar to the standard BP iterations, we define $\mathbf{M}^{V,l}$ and $\mathbf{M}^{C,l}$ as the matrices consisting of $\mu_{v,c}^{(l)}$'s (messages sent from variable nodes to check nodes in the lth iteration) and $\mu_{c,v}^{(l)}$'s (messages sent from check nodes to variable nodes in the lth iteration), respectively. In particular, entry $[\mathbf{M}^{V,l}]_{i,j}$ (alternatively, $[\mathbf{M}^{C,l}]_{i,j}$) is equal to the message $\mu_{v_j,c_i}^{(l)}$ (alternatively, $\mu_{c_i,v_j}^{(l)}$). Hence, we can rewrite (2) as follows:

$$[\mathbf{M}^{V,l}]_{i,j} = \begin{cases} 1, & [\mathbf{H}]_{i,j} = 0, \\ [\mathbf{M}]_{i,j} + \sum\limits_{t=1, t\neq i}^{n-k} [\mathbf{M}^{C,l-1}]_{t,j}, & [\mathbf{H}]_{i,j} \neq 0, \end{cases} \tag{5}$$

where $\mathbf{M}^{C,-1}$ is the zero matrix of size $(n-k)\times n$. Similarly, we rewrite (3) in the following form:

$$[\mathbf{M}^{C,l}]_{i,j} = \begin{cases} 0, & [\mathbf{H}]_{i,j} = 0, \\ 2\text{Arctanh}\left(\prod\limits_{t=1, t\neq j}^{n} \tanh\left(\frac{[\mathbf{M}^{V,l}]_{i,t}}{2}\right)\right), & [\mathbf{H}]_{i,j} \neq 0. \end{cases} \tag{6}$$

Finally, the equivalent of (4) for the output of variable node v_j at the lth iteration is given by:

$$[\mathbf{O}^l]_j = \mu_j + \sum_{t=1}^{n-k} [\mathbf{M}^{C,l}]_{t,j}. \tag{7}$$

B. Neural Decoder based on Weighted BP Algorithm

Here, we interpret the introduced matrix form representation of the BP algorithm as the forward passes in a neural network. More specifically, the entries of $\mathbf{M}^{C,l}$ can be interpreted as neurons of the lth hidden layer in a neural network: some linear combinations of these neurons are computed (see eq. (5)) and passed through a non-linear function (see eq. (6)) to generate the values of the neurons in the next layer. In the standard BP algorithm, the entries of $\mathbf{M}^{C,l}$ shall be simply combined with equal unit weights. However, a more careful weighting scheme could considerably improve the performance of decoder. Thus, we propose a neural decoder (called "Mat-Net"), which combines entries of $\mathbf{M}^{C,l}$ by some weights. We adjust these weights by training the neural network. Concretely, for each layer l, we introduce a weight matrix $\mathbf{W}^{V,l}$ and a bias matrix $\mathbf{B}^{V,l}$ as follows:

$$[\mathbf{W}^{V,l}]_{i,j} = \begin{cases} 0, & [\mathbf{H}]_{i,j} = 0, \\ w_{i,j}^{V,l}, & [\mathbf{H}]_{i,j} \neq 0, \end{cases} \tag{8}$$

$$[\mathbf{B}^{V,l}]_{i,j} = \begin{cases} 0, & [\mathbf{H}]_{i,j} = 0, \\ b_{i,j}^{V,l}, & [\mathbf{H}]_{i,j} \neq 0, \end{cases} \tag{9}$$

to form $\tilde{\mathbf{M}}^{C,l-1} = \mathbf{W}^{V,l} \circ \mathbf{M}^{C,l-1}$ and $\tilde{\mathbf{M}}^{l} = \mathbf{B}^{V,l} \circ \mathbf{M}$. Now, (5) is modified to

$$[\mathbf{M}^{V,l}]_{i,j} = \begin{cases} 1, & [\mathbf{H}]_{i,j} = 0, \\ [\tilde{\mathbf{M}}^{l}]_{i,j} + \sum\limits_{t=1, t\neq i}^{n-k} [\tilde{\mathbf{M}}^{C,l-1}]_{t,j}, & [\mathbf{H}]_{i,j} \neq 0. \end{cases} \tag{10}$$

We should add that (6) remains intact. Finally, by introducing the weight matrix $\mathbf{W}^{O,l}$ as

$$[\mathbf{W}^{O,l}]_{i,j} = \begin{cases} 0, & [\mathbf{H}]_{i,j} = 0, \\ w_{i,j}^{O,l}, & [\mathbf{H}]_{i,j} \neq 0, \end{cases} \tag{11}$$

and the vector $\mathbf{B}^{O,l}$ of size n, we first update $\mathbf{M}^{C,l}$ and $\boldsymbol{\mu}$ via

$$\begin{aligned} \tilde{\mathbf{M}}^{C,l} &= \mathbf{W}^{O,l} \circ \mathbf{M}^{C,l}, \\ \tilde{\boldsymbol{\mu}} &= \mathbf{B}^{O,l} \circ \boldsymbol{\mu}, \end{aligned} \tag{12}$$

and then compute the output at each layer l by

$$[\mathbf{O}^l]_j = \tilde{\mu}_j + \sum_{t=1}^{t=n-k} [\tilde{\mathbf{M}}^{C,l}]_{t,j}. \tag{13}$$

We consider the following loss function:

$$F_s = -\frac{1}{N} \sum_{i=s}^{L} \sum_{j=1}^{n} x_j \log\left(o_j^i\right) + \left(1 - x_j\right) \log\left(1 - o_j^i\right), \tag{14}$$

where x_j is the true value of the jth bit sent over the channel and o_j^i is the output of the ith iteration for the jth bit. We can consider the output of all iterations in the loss function by setting $s = 1$ or just the output of the last iteration by setting $s = L$, where L is the number of iterations.

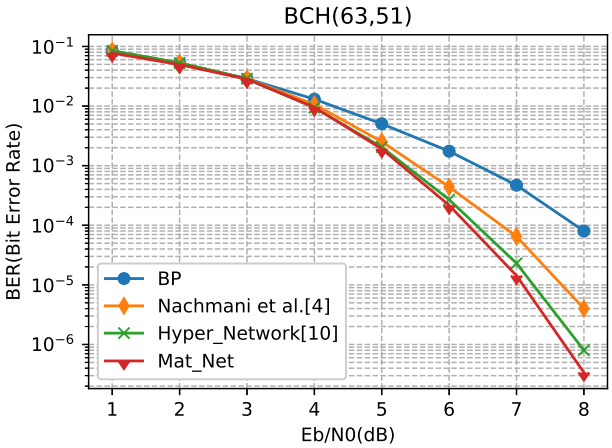

Fig. 1. The BER of different decoders for the BCH(63, 51) code.

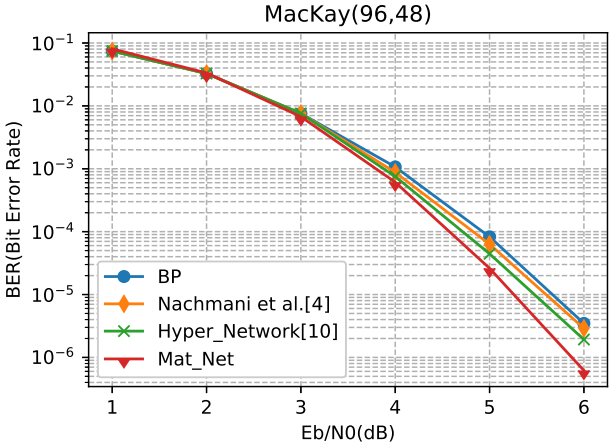

Fig. 2. The BER of different decoders for MacKay(96, 48) code.

IV. Experiment and Numerical Results

We evaluate the performance of the proposed neural decoder on three types of codes: LDPC [18], Polar [19] and BCH [20]. The Adam optimizer [21] is utilized with learning rate of 0.001 for Polar and BCH(127, 64) codes, and with learning rate of 0.0001 for LDPC and BCH(63, 51) codes. The proposed network and other decoding algorithms are compared with $L = 5$ iterations and all parity check matrices are taken from [22]. For generating the training data, zero code-word was sent over AWGN channel and received vectors were collected[1]. In all experiments, we used the BPSK modulation. Moreover, the size of the mini-batch is set to 200 and each mini-batch consists of 20 code-words per SNR. In the training phase, the selected SNRs for Polar and LDPC codes are $\{4, 4.5, 5, .., 8.5\}$dB and for BCH codes are $\{5, 5.5, 6, .., 9.5\}$. The parameter s in the loss function is set to L for BCH and LDPC codes whereas $s = 1$ is implemented for Polar codes. Source code of the proposed solution is available online [23].

[1]We can train the proposed network by merely zero code-word since it satisfies the symmetry property in [5].

TABLE I
THE NEGATIVE NATURAL LOGARITHMS OF BER IN THREE SNRS OF 4, 5, 6 DB. HIGHER IS BETTER.

method	BP			[4]			[10]			Mat-Net		
SNR	4	5	6	4	5	6	4	5	6	4	5	6
BCH(127,64)	2.99	3.60	4.29	2.99	3.79	4.74	–	–	–	3.00	3.86	5.19
LDPC(121,70)	5.88	8.76	13.04	6.27	9.44	13.47	6.39	9.81	14.04	6.53	10.25	14.88
Polar(64,48)	4.15	4.68	5.31	4.77	6.12	7.84	4.91	6.48	8.41	4.96	6.58	8.55
Polar(128,96)	3.99	4.41	4.78	4.56	5.98	7.53	4.73	6.39	8.57	4.76	6.42	8.64

TABLE II
THE SECOND, THIRD, AND FIFTH COLUMNS SHOW THE NUMBER OF MULTIPLICATIONS IN OUR METHOD, [4], AND [10], RESPECTIVELY.

method	Mat-Net	[4]		[10]	
BCH(63,51)	58191	63388	8%	812208	93%
BCH(127,64)	447805	654728	32%	–	–
MacKay(96,48)	11904	11808	-0.8%	582368	98%
LDPC(121,70)	43076	49005	12%	1270185	97%
Polar(64,48)	77328	87144	11%	1035184	93%
Polar(128,96)	401200	467112	14%	3810512	89%

Figures 1 and 2 show the Bit Error Rate (BER) curves for BCH$(63, 51)$ and LDPCMacKay$(96, 48)$ codes for input SNRs in the range $[1, 8]$ and $[1, 6]$, respectively. We compared our network with the BP algorithm and the most relevant methods in [4], [10]. The results show that our method improves the BER up to 0.4dB for BCH$(63, 51)$ and 0.18dB for MacKay$(96, 48)$ compared to the best previous method [10]. Table I provides the negative natural logarithm of the BER of four other codes for SNR$= 4, 5, 6$ dB. As can be seen, our method performs better than [4], [10] and the BP algorithm in all these codes.

The number of multiplications in a decoder have a significant effect on the implementation cost. Thus, we compared different methods in terms of the number of multiplications. Table II shows the number of multiplications in networks of [4], [10], and our method. The percentages in the forth and sixth columns show the amount of improvement in the number of multiplications with respect to [4] and [10], respectively. As can be seen, our network reduces the number of multiplications up to 32% and 98% with respect to [4] and [10], respectively.

V. CONCLUSION

The common implementation of the BP algorithm is based on the Tanner graph. In this work, we proposed a matrix form representation of the BP algorithm which directly deals with the parity check matrix. Based on this representation, we proposed a neural network by introducing some weights for the links of the neural network. Simulation results showed that the proposed network outperforms previous works in terms of BER and also reduces the computational complexity in terms of the number of multiplications.

REFERENCES

[1] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2017, pp. 1–6.

[2] S. Cammerer, T. Gruber, J. Hoydis, and S. Ten Brink, "Scaling deep learning-based decoding of polar codes via partitioning," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[3] E. Arıkan, "Polar codes: A pipelined implementation," in *Proc. 4th Int. Symp. on Broad. Commun. ISBC 2010*, 2010, pp. 11–14.

[4] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. BeâĂŹery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[5] T. Richardson and R. Urbanke, *Modern coding theory*. Cambridge university press, 2008.

[6] W. Xu, Z. Wu, Y.-L. Ueng, X. You, and C. Zhang, "Improved polar decoder based on deep learning," in *2017 IEEE International workshop on signal processing systems (SiPS)*. IEEE, 2017, pp. 1–6.

[7] L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 1361–1365.

[8] B. Vasić, X. Xiao, and S. Lin, "Learning to decode ldpc codes with finite-alphabet message passing," in *2018 Information Theory and Applications Workshop (ITA)*. IEEE, 2018, pp. 1–9.

[9] S. Malek, S. Salehkaleybar, and A. Amini, "Multi variable-layer neural networks for decoding linear codes," in *2020 Iran Workshop on Communication and Information Theory (IWCIT)*. IEEE, 2020, pp. 1–6.

[10] E. Nachmani and L. Wolf, "Hyper-graph-network decoders for block codes," in *Advances in Neural Information Processing Systems*, 2019, pp. 2326–2336.

[11] ——, "A gated hypernet decoder for polar codes," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 5210–5214.

[12] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," in *Sixth International Conference on Learning Representations (ICLR)*, 2018.

[13] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Joint channel coding and modulation via deep learning," in *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2020, pp. 1–5.

[14] ——, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," *Advances in neural information processing systems*, vol. 32, pp. 2758–2768, 2019.

[15] S. Habib, A. Beemer, and J. Kliewer, "Learning to decode: Reinforcement learning for decoding of sparse graph-based channel codes," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[16] X. Ni, "Neural network decoders for large-distance 2d toric codes," *Quantum*, vol. 4, p. 310, 2020.

[17] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[18] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[19] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[20] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of Channel Codes and ML Simulation Results," www.uni-kl.de/channel-codes, 2019.

[23] [Online]. Available: https://github.com/SamiraMalek/MatNet.git